Programming Project #5: Video Stitching and Processing

CS445: Computational Photography - Fall 2019

Part I: Stitch two key frames

This involves:

1. compute homography H between two frames;
2. project each frame onto the same surface;
3. blend the surfaces.

Check that your homography is correct by plotting four points that form a square in frame 270 and their projections in each image, like this:

In [1]:
```
import cv2
import numpy as np
from numpy.linalg import svd, inv, solve, lstsq
import scipy
from scipy.sparse.linalg import lsqr
from scipy.sparse import csr_matrix
%matplotlib inline
from matplotlib import pyplot as plt
```

In [2]:
```
# images location
im1 = './images/input/frames/f0001.jpg'
im2 = './images/input/frames/f0270.jpg'

# Load an color image in grayscale
im1 = cv2.imread(im1)
im2 = cv2.imread(im2)
```

In [3]:

```
def auto_homography(Ia,Ib, homography_func=None, normalization_func=None):
    '''
    Computes a homography that maps points from Ia to Ib

    Input: Ia and Ib are images
    Output: H is the homography

    '''
    if Ia.dtype == 'float32' and Ib.dtype == 'float32':
        Ia = (Ia*255).astype(np.uint8)
        Ib = (Ib*255).astype(np.uint8)

    Ia_gray = cv2.cvtColor(Ia,cv2.COLOR_BGR2GRAY)
    Ib_gray = cv2.cvtColor(Ib,cv2.COLOR_BGR2GRAY)

    # Initiate SIFT detector
    sift = cv2.xfeatures2d.SIFT_create()

    # find the keypoints and descriptors with SIFT
    kp_a, des_a = sift.detectAndCompute(Ia_gray,None)
    kp_b, des_b = sift.detectAndCompute(Ib_gray,None)

    # BFMatcher with default params
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des_a,des_b, k=2)

    # Apply ratio test
    good = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append(m)

    numMatches = int(len(good))

    matches = good

    # Xa and Xb are 3xN matrices that contain homogeneous coordinates for the
    # matching points for each image
    Xa = np.ones((3,numMatches))
    Xb = np.ones((3,numMatches))

    for idx, match_i in enumerate(matches):
        Xa[:,idx][0:2] = kp_a[match_i.queryIdx].pt
        Xb[:,idx][0:2] = kp_b[match_i.trainIdx].pt

    ## RANSAC
    niter = 1000
    best_score = 0

    for t in range(niter):
        # estimate homography
        subset = np.random.choice(numMatches, 4, replace=False)
        pts1 = Xa[:,subset]
        pts2 = Xb[:,subset]

        H_t = homography_func(pts1, pts2, normalization_func) # edit helper c
```

```
57          #H_t = computeHomography(pts1, pts2)
58
59          # score homography
60          Xb_ = np.dot(H_t, Xa) # project points from first image to second usi
61          du = Xb_[0,:]/Xb_[2,:] - Xb[0,:]/Xb[2,:]
62          dv = Xb_[1,:]/Xb_[2,:] - Xb[1,:]/Xb[2,:]
63
64          ok_t = np.sqrt(du**2 + dv**2) < 1  # you may need to play with this t
65          score_t = sum(ok_t)
66
67          if score_t > best_score:
68              print("get a better score")
69              best_score = score_t
70              H = H_t
71              in_idx = ok_t
72
73      print('best score: {:02f}'.format(best_score))
74
75      # Optionally, you may want to re-estimate H based on inliers
76
77      return H
```

In [4]:

```
1  from scipy import linalg
2
3  def get_A(src,dst):
4      eye = np.eye(3)
5      zeros = np.zeros([3,3])
6      first_half = np.concatenate((eye*-1,zeros,dst[0]*eye), axis=-1)
7      second_half = np.concatenate((zeros,eye*-1,dst[1]*eye), axis=-1)
8      #print(src.T.dot(first_half).shape, src.T.dot(second_half).shape)
9      return src.T.dot(first_half),src.T.dot(second_half)
10
11 def computeHomography(pts1, pts2, normalization_func=None):
12     '''
13     Compute homography that maps from pts1 to pts2 using least squares solver
14
15     Input: pts1 and pts2 are 3xN matrices for N points in homogeneous
16     coordinates.
17
18     Output: H is a 3x3 matrix, such that pts2~=H*pts1 pts1.T * H^T = pts2.T
19     '''
20     N = pts1.shape[1]
21     l = []
22     for i in range(N):
23         a,b = get_A(pts1[:,i],pts2[:,i])
24         l.append(a)
25         l.append(b)
26     #print(l)
27     A = np.asarray(l)
28     #print(A.shape)
29     #print(A)
30     U,s,Vt = np.linalg.svd(A)
31     #print(Vt.shape)
32     H = Vt[-1,:].reshape([3,3])
33     return H
34
```

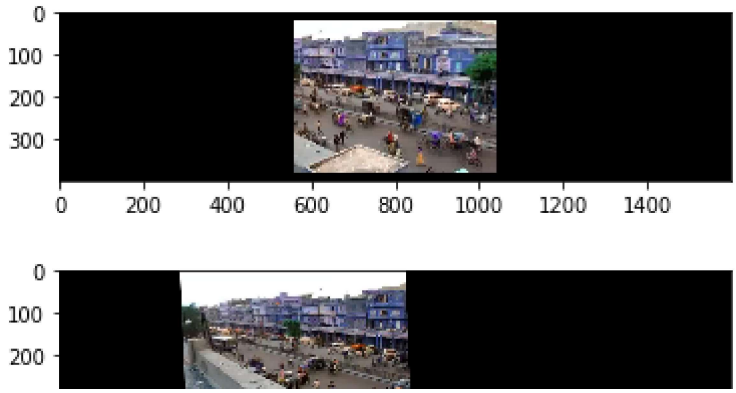
In [5]:
```
1 H = auto_homography(im1,im2, computeHomography)
```

```
get a better score
get a better score
get a better score
get a better score
get a better score
get a better score
get a better score
best score: 159.000000
```

In [6]:

```
output_width, output_height = (1600, 400)

corners = np.array([np.array([[0,0],[im1.shape[1]-1,0],[0,im1.shape[0]-1],[im
center = np.array([np.array([[output_width//2-im1.shape[1]//2, output_height/
                             [output_width//2+im1.shape[1]//2, output_height/
                             [output_width//2-im1.shape[1]//2, output_height/
                             [output_width//2+im1.shape[1]//2, output_height/
#print(corners)
H_t = cv2.getPerspectiveTransform(corners,center)
print(H_t)
shift = np.array([[1,0,800],[0,1,0],[0,0,1]]).astype(np.float32)
shift2 = np.array([[0,0,800],[0,0,0],[0,0,0]]).astype(np.float32)
img_warped = cv2.warpPerspective(im2, H_t, (output_width, output_height)).ast
plt.imshow(img_warped/255.)
plt.show()

img_warped2 = cv2.warpPerspective(im1, H_t.dot(H), (output_width, output_heig
mask = np.where(img_warped>0, 1, 0).astype(np.float32)
plt.imshow(img_warped2/255.)
plt.show()

print(np.max(img_warped))
print("mask")
plt.imshow(mask.astype(np.float32))
plt.show()
print("background")
plt.imshow(img_warped2.astype(np.float32)/255.)
plt.show()
print("foreground")
plt.imshow(img_warped.astype(np.float32)/255.)
plt.show()
print(np.sum(mask,axis=-1).shape)
#img_warped3 = laplacian_blend(img_warped, img_warped2, np.sum(mask,axis=-1))
#img_warped3 = mask*img_warped+(1.-mask)*img_warped2
#alpha blend
img_warped3 = mask*img_warped+(1.-mask)*img_warped2

result = cv2.cvtColor(img_warped3,cv2.COLOR_BGR2RGB)

print(H)
plt.imshow(result/255.)
plt.show()
print(np.max(result))
plt.imsave("trial1.jpg",np.clip(result/255.,0.,1.))
del result
del img_warped, img_warped2, img_warped3
```

```
[[ 1.00208768e+00 -6.66133815e-16  5.60000000e+02]
 [-1.33226763e-15  1.00278552e+00  2.00000000e+01]
 [-1.73472348e-18  4.33680869e-19  1.00000000e+00]]
```



Part II: Panorama using five key frames

In this part you will produce a panorama using five key frames. Let's determine frames [90, 270, 450, 630, 810] as key frames. The goal is to map all the five frames onto the plane corresponding to frame 450 (that we also call the *reference frame*). For the frames 270 and 630 you can follow the instructions in part 1.

Mapping frame 90 to frame 450 is difficult because they share very little area. Therefore you need to perform a two stage mapping by using frame 270 as a guide. Compute one projection from 90 to 270 and one from 270 to 450 and multiply the two matrices. This produces a projection from 90 to 450 even though these frames have very little area in common

In [7]:
```
1 import cv2
2 import numpy as np
```

In [8]:

```
master_frames =[90, 270, 450, 630, 810]
output_width, output_height = 1600,500
img_list = []
img_path_list=['./images/input/frames/f%04d.jpg'%k for k in master_frames]
print(img_path_list)
img_list = [cv2.imread(t) for t in img_path_list]

reference_frame = 450
reference_idx = master_frames.index(reference_frame)
H_list = []
corners = np.array([np.array([[0,0],[im1.shape[1]-1,0],[0,im1.shape[0]-1],[im
center = np.array([np.array([[output_width//2-im1.shape[1]//2, output_height/
                             [output_width//2+im1.shape[1]//2, output_height/
                             [output_width//2-im1.shape[1]//2, output_height/
                             [output_width//2+im1.shape[1]//2, output_height/
H_t = cv2.getPerspectiveTransform(corners,center)
shift = np.array([[1,0,output_width//2-img_list[reference_idx].shape[1]//2],[
shift2 = np.array([[0,0,output_width//2-img_list[reference_idx].shape[1]//2],
for i in range(reference_idx):
    H_list.append(auto_homography(img_list[i],img_list[i+1], computeHomograph
for i in range(reference_idx+1, len(master_frames)):
    H_list.append(auto_homography(img_list[i],img_list[i-1], computeHomograph

img_warped = []
mask = []
for i in range(reference_idx):
    #print("image: ",i)
    H = H_list[i]
    #print(H)
    for j in range(reference_idx):
        if j>i:
            print(H_list[j])
            H = np.dot(H_list[j],H)
    #print("H: ")
    #print(H)
    img_warped.append(cv2.warpPerspective(img_list[i], H_t.dot(H), (output_wi
    mask.append(np.where(img_warped[-1]>0,1.,0.))

img_warped.append(cv2.warpPerspective(img_list[reference_idx], H_t, (output_w
mask.append(np.where(img_warped[-1]>0,1.,0.))

for i in range(reference_idx+1, len(master_frames)):
    H = H_list[i-1]
    for j in range(i-2,0,-1):
        if j>=reference_idx:
            H = np.dot(H_list[j],H)
    img_warped.append(cv2.warpPerspective(img_list[i], H_t.dot(H), (output_wi
    mask.append(np.where(img_warped[-1]>0,1.,0.))


result = np.zeros((output_height,output_width,3))
for i in range(len(master_frames)-1,-1,-1):
    #print("result: ",result.shape)
    #print("mask: ",mask[i].shape)
    #print("img: ",img_list[i].shape)
    result = result*(1-mask[i])+img_warped[i]
```

```
57
58 result = cv2.cvtColor(np.clip(result.astype(np.uint8),0,255),cv2.COLOR_BGR2RG
59 plt.imshow(np.clip(result/255.,0.,1.))
60 plt.show()
61 plt.imsave("panoramic.jpg",result)
62 #img_warped3 = laplacian_blend(img_warped, img_warped2, np.sum(mask,axis=-1))
63 #img_warped3 = mask*img_warped+(1.-mask)*img_warped2
64 #alpha blend
65 #img_warped3 = mask*img_warped+(1.-mask)*img_warped2
66
67 #result = cv2.cvtColor(img_warped3,cv2.COLOR_BGR2RGB)
68 del img_list, result, img_warped
```

```
['./images/input/frames/f0090.jpg', './images/input/frames/f0270.jpg', './ima
ges/input/frames/f0450.jpg', './images/input/frames/f0630.jpg', './images/inp
ut/frames/f0810.jpg']
get a better score
get a better score
get a better score
get a better score
best score: 210.000000
get a better score
get a better score
get a better score
get a better score
get a better score
get a better score
best score: 158.000000
get a better score
get a better score
get a better score
get a better score
```

Part 3: Map the video to the reference plane

In [9]:
```
1 import os
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from math import floor
6
7 import utils
```

In [10]:
```
1 dir_frames = 'images/input/frames'
2 filenames = []
3 filesinfo = os.scandir(dir_frames)
```

In [11]:
```
1 filenames = [f.path for f in filesinfo if f.name.endswith(".jpg")]
2 filenames.sort(key=lambda f: int(''.join(filter(str.isdigit, f))))
```

In [12]:
```
frameCount = len(filenames)
frameHeight, frameWidth, frameChannels = cv2.imread(filenames[0]).shape
frames = np.zeros((frameCount, frameHeight, frameWidth, frameChannels),dtype=
```

In [13]:
```
for idx, file_i in enumerate(filenames):
    frames[idx] = cv2.cvtColor(cv2.imread(file_i), cv2.COLOR_BGR2RGB) / 255.0
```

In [28]:

```
## Example usage of utils.projectImage
H = 700
W = 2000
#pastHomographies = np.zeros((len(filenames),len(filenames), 3, 3),dtype=np.f
originTranslations = np.zeros((len(filenames), 2), dtype=np.float32)
originTranslations[:,0]-=1000-int(frames[0].shape[1]/2)
originTranslations[:,1]-=350-int(frames[0].shape[0]/2)
#print(originTranslations[451:])
#originTranslations[451:,0]+=1000-int(frames[0].shape[1]/2)#1000-int(frames[0
#originTranslations[451:,1]+=350-int(frames[0].shape[0]/2)#350-int(frames[0].
referenceFrameIndex = 450

for i in range(450):#len(frames)):
    sourceFrameIndex = int(i)
    print(i)
    #print(len(frames))
    #print(frames[0].shape)
    projectedSource, pastHomographies, originTranslations = utils.projectImag
                                                            pastHomograph
                                                            yrange=H, ove
                                                            numKeyframes=
                                                            auto_H_func=a
    #plt.imshow(projectedSource)
    #plt.show()
    #print(past_trans1.shape)
    name = cv2.imwrite('aligned_frames/a{:04d}.jpg'.format(i), cv2.cvtColor(p
    #trans_map[i,:,:]=past_trans1[:,:]
'''


for i in range(frameCount-1,450,-1):#len(frames)):
    sourceFrameIndex = int(i)
    print(i)
    #print(len(frames))
    #print(frames[0].shape)
    projectedSource, pastHomographies, originTranslations = utils.projectImag
                                                            pastHomograph
                                                            yrange=H, ove
                                                            numKeyframes=
                                                            auto_H_func=a
    #plt.imshow(projectedSource)
    #plt.show()
    name = cv2.imwrite('aligned_frames/a{:04d}.jpg'.format(i), cv2.cvtColor(p
    #trans_map[i,:,:]=past_trans2[:,:]
'''
np.save("trans",pastHomographies)
    #utils.imageFolder2mpeg('aligned_frames', fps=30)
```

```
0
Overlap:169200
Error:0.000493949100987447
Finding better homography...
Overlap:135386
Error:0.0005839388238521869
Overlap:148966
```

```
Error:0.00018655811193993158
Overlap:67335
Error:0.0007513987659576506
get a better score
get a better score
get a better score
get a better score
get a better score
best score: 154.000000
Overlap:106438
Error:0.00027355704899158716
Overlap:5408
```

In [29]:
```
1 utils.imageFolder2mpeg('aligned_frames', fps=30)
```

Part 4: Create background panorama

In this part you will remove moving objects from the video and create a background panorama that should incorporate pixels from all the frames.

In the video you produced in **part 3** each pixel appears in several frames. You need to estimate which of the many colors correspond to the background. We take advantage of the fact that the background color is fixed while the foreground color changes frequently (because foreground moves).

For each pixel in the sequence of **part 3**, determine all valid colors (colors that come from all frames that overlap that pixel). You can experiment with different methods for determining the background color of each pixel, as discussed in class. Perform the same procedure for all pixels and generate output. The output should be a completed panorama showing only pixels of background or non-moving objects.

In [30]:
```
1 import os
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
```

In [34]:
```
## Example usage of utils.projectImage
H = 512
W = 1632
mean = np.zeros((H,W,3))
count = np.zeros((H,W,3))
print(mean.shape)

filenames = [f.path for f in os.scandir('./aligned_frames/') if f.name.endswi
#print(filenames)
filenames.sort(key=lambda f: int(f[len('./aligned_frames/')+1:len('./aligned_
print(filenames)
frame_list = [plt.imread(f) for f in filenames]
num_frames = len(frame_list)
```

```
(512, 1632, 3)
['./aligned_frames/a0000.jpg', './aligned_frames/a0001.jpg', './aligned_frame
s/a0002.jpg', './aligned_frames/a0003.jpg', './aligned_frames/a0004.jpg', './
aligned_frames/a0005.jpg', './aligned_frames/a0006.jpg', './aligned_frames/a0
007.jpg', './aligned_frames/a0008.jpg', './aligned_frames/a0009.jpg', './alig
ned_frames/a0010.jpg', './aligned_frames/a0011.jpg', './aligned_frames/a0012.
jpg', './aligned_frames/a0013.jpg', './aligned_frames/a0014.jpg', './aligned_
frames/a0015.jpg', './aligned_frames/a0016.jpg', './aligned_frames/a0017.jp
g', './aligned_frames/a0018.jpg', './aligned_frames/a0019.jpg', './aligned_fr
ames/a0020.jpg', './aligned_frames/a0021.jpg', './aligned_frames/a0022.jpg',
'./aligned_frames/a0023.jpg', './aligned_frames/a0024.jpg', './aligned_frame
s/a0025.jpg', './aligned_frames/a0026.jpg', './aligned_frames/a0027.jpg', './
aligned_frames/a0028.jpg', './aligned_frames/a0029.jpg', './aligned_frames/a0
030.jpg', './aligned_frames/a0031.jpg', './aligned_frames/a0032.jpg', './alig
ned_frames/a0033.jpg', './aligned_frames/a0034.jpg', './aligned_frames/a0035.
jpg', './aligned_frames/a0036.jpg', './aligned_frames/a0037.jpg', './aligned_
frames/a0038.jpg', './aligned_frames/a0039.jpg', './aligned_frames/a0040.jp
g', './aligned_frames/a0041.jpg', './aligned_frames/a0042.jpg', './aligned_fr
ames/a0043.jpg', './aligned_frames/a0044.jpg', './aligned_frames/a0045.jpg',
' ./aligned_frames/a0046.jpg', ' ./aligned_frames/a0047.jpg', ' ./aligned_frame
```

In [35]:
```
#del frames
for t in range(len(frame_list)):
    if frame_list[t].shape!=(H,W,3):
        print(t,frame_list[t].shape)
```

In [36]:

```
frame_list = np.concatenate(frame_list,axis=-1).reshape((H,W,frameCount,3))

frame_color_R = np.zeros((H,W,frameCount))
frame_color_R += frame_list[:,:,:,0]
frame_color_G = np.zeros((H,W,frameCount))
frame_color_G += frame_list[:,:,:,1]
frame_color_B = np.zeros((H,W,frameCount))
frame_color_B += frame_list[:,:,:,2]
del frame_list
R = np.zeros((H,W))
frame_color_R = np.sort(frame_color_R,axis=-1)
for y in range(H):
    print(y)
    for x in range(W):
        count = -1
        while frame_color_R[y,x,count]:
            count-=1;
        R[y,x] = np.median(frame_color_R[y,x,count:])
del frame_color_R

G = np.zeros((H,W))
frame_color_G = np.sort(frame_color_G,axis=-1)
for y in range(H):
    print(y)
    for x in range(W):
        count = -1
        while frame_color_G[y,x,count]:
            count-=1;
        G[y,x] = np.median(frame_color_G[y,x,count:])
del frame_color_G

B = np.zeros((H,W))
frame_color_B = np.sort(frame_color_B,axis=-1)
for y in range(H):
    print(y)
    for x in range(W):
        count = -1
        while frame_color_B[y,x,count]:
            count-=1;
        B[y,x] = np.median(frame_color_B[y,x,count:])
del frame_color_B

result = cv2.merge([R,G,B])


#plt.imshow(frame_list[:,:,0,:].reshape((H,W,3)))
# =
#color_map =
```

```
0
1
2
3
4
5
6
```

```
7
8
9
10
11
12
13
14
15
16
17
18
```

In [37]:
```
1
2
3 print(result.shape)
4 print(np.max(result))
5 plt.imshow(result.astype(np.uint8))
6 plt.show()
7
8 plt.imsave("median_background.jpg",(result).astype(np.uint8))
```

```
(512, 1632, 3)
255.0
```

Part 5: Create background movie

Map the background panorama to the movie coordinates. For each frame of the movie, say frame 1, you need to estimate a projection from the panorama to frame 1. Note, you should be able to re-use the homographies that you estimated in **Part 3**. Perform this for all frames and generate a movie that looks like the input movie but shows only background pixels. All moving objects that belong to the foreground must be removed.

In [38]:
```
1 import os
2 import cv2
3 import numpy as np
4 pastHomographies = np.load("trans.npy")
5 result = plt.imread("median_background.jpg")
```

In [39]:
```
## Example usage of utils.projectImage

mean = np.zeros((H,W,3))
count = np.zeros((H,W,3))
print(mean.shape)

filenames = [f.path for f in os.scandir('./aligned_frames/') if f.name.endswi
#print(filenames)
filenames.sort(key=lambda f: int(f[len('./aligned_frames/')+1:len('./aligned_
#print(filenames)
frame_list = [plt.imread(f) for f in filenames]
for i in range(len(frame_list)):
    frame = frame_list[i]
    #plt.imshow(frame)
    #plt.show()
    #foreground_mask = np.where(np.abs(frame-median)>thredhold, 1, 0)
    background_mask = np.where(frame,1.,0.)
    #plt.imshow(background_mask)
    #plt.show()
    #print(background_mask.shape)
    print(i)
    #print(pastHomographies[i,450])
    H = np.linalg.inv(pastHomographies[i,450])
    background_frame = result*background_mask
    warped = cv2.warpPerspective(background_frame, H, (480,360)).astype(np.ui
    #plt.imshow(background_frame/255.)
    #plt.show()
    name = plt.imsave('aligned_background_frames/a{:04d}.jpg'.format(i), warp

```

```
(512, 1632, 3)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

In [41]:
```
utils.imageFolder2mpeg('aligned_background_frames','./output_background_video
```

Part 6: Create foreground movie

In the background video, moving objects are removed. In each frame, those pixels that are different enough than the background color are considered foreground. For each frame determine foreground pixels and generate a movie that only includes foreground pixels.

In [44]:
```
import os
import cv2
import numpy as np

## Example usage of utils.projectImage
print(mean.shape)

filenames = [f.path for f in os.scandir('./aligned_frames/') if f.name.endswi
#print(filenames)
filenames.sort(key=lambda f: int(f[len('./aligned_frames/')+1:len('./aligned_
#print(filenames)
frame_list = [plt.imread(f) for f in filenames]
for i in range(len(frame_list)):
    frame = frame_list[i]
    print(i)
    background_mask = np.where(frame,1.,0.)
    background_frame = result*background_mask

    dist = np.sum((frame-background_frame)**2,axis=-1)
    #print(dist.shape)
    dist = cv2.merge([dist,dist,dist])
    foreground_frame_mask = np.where(dist>2000, 1, 0)
    #print(foreground_frame_mask.shape)
    foreground_frame = frame*foreground_frame_mask
    H = np.linalg.inv(pastHomographies[i,450])
    warped = cv2.warpPerspective(foreground_frame.astype(np.uint8), H, (480,3
    name = plt.imsave('aligned_foreground_frames/a{:04d}.jpg'.format(i), warp

```

```
(512, 1632, 3)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

In [45]:
```
utils.imageFolder2mpeg('aligned_foreground_frames','./output_foreground_video
```

Bells and whistles

In [ ]:
```
1
```